FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Automatic Parallelization of a Crystal Growth Simulation Program for Distributed-Memory Systems

Michael Gerndt

KFA-ZAM-IB-9404

Februar 1994
(Stand 17.02.94)

To be published in the proceedings of HPCN Europe, April 19-20, 1994

Automatic Parallelization of a Crystal Growth Simulation Program for Distributed-Memory Systems

Michael Gerndt

Central Institute for Applied Mathematics
Research Centre Jülich
D-52425 Jülich
email: m.gerndt@kfa-juelich.de

Abstract. This article outlines two parallelization tools, i.e. the Vienna Fortran Compilation System and FORGE 90, and discusses their analysis and transformation capabilities in the context of a regular grid application simulating the growth of a silicon crystal. We present performance results obtained on an iPSC/860 for both versions and for a manually parallelized version.

1 Introduction

Parallelization of sequential applications for massively parallel systems is currently performed manually. The programmer first has to rewrite the whole application in the message passing programming model before test runs can be performed to evaluate the parallelization strategy. Manual parallelization can be facilitated by appropriate tools, such as TOP^2 [1] that was developed at KFA and allows the programmer to separate individual subroutines and provides an environment for test runs of the parallel implementation of that subroutine.

Automatic parallelization is supported only by a few tools, e.g. the Vienna Fortran Compilation System (VFCS) [6, 4], the Fortran D environment [3], and FORGE 90, xHPF77 [2]. All these tools are based on the data distribution approach. The user specifies the distribution of arrays to the processors and the parallelization tool adapts the sequential code to the specified distribution. The annotations specifying the data distribution have been standardized as High Performance Fortran (HPF) [5].

In this article we introduce two tools, the VFCS and FORGE 90. We parallelized a regular grid application simulating the melting process of silicon with both tools and also used a hand-coded parallel version developed according to the same parallelization strategy to analyze the effectiveness of the parallelization tools.

2 Crystal Growth Simulation

The Crystal Growth Simulation is an application developed at KFA [7] for the optimization of the silicon production process. For the quality of the silicon

crystal a constant convection in the melt is very important. The convection results from the heating, the rotation of the crucible, and the rotation of the crystal. The convection is modeled by a set of partial differential equations and determined by an explicit finite difference scheme. Thus, array references are regular and the application is a good candidate for state-of-the-art parallelization tools.

The simulated crucible has a radius of 3 cm and a height of 4 cm and is discretized into 30 x 90 x 40 elements. This determines the shape of the main arrays to be 32 x 92 x 42 including some additional boundary cells. The boundary cells determine the boundary conditions at the surface of the crucible, e.g. the temperature of the heating, and the values at the inner surface where the crucible is unfolded to give a regular three-dimensional structure. This relation between the crucible and the main arrays is shown in Figure 1.

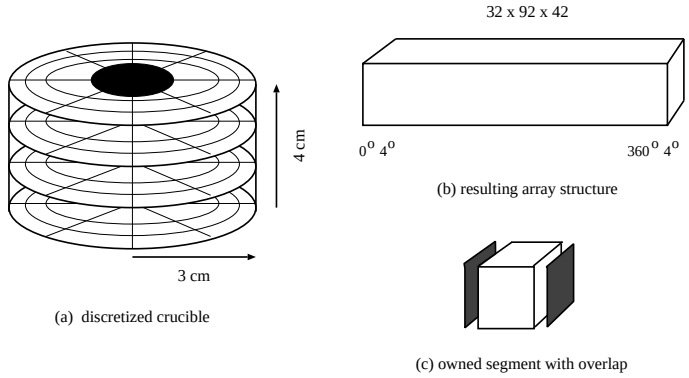

Fig. 1. Application Domain

The algorithm consists of an initialization phase, a time loop, and an output phase. In the time loop for each time step the new temperature, the pressure, and the velocity are computed and the boundary conditions are updated. The most time consuming procedure is the computation of the velocity and the pressure. Here the linear equation system resulting from the partial differential equations is solved by successive overrelaxation.

In this subroutine as well as in the other operations of the time loop mostly stencil operations are performed on the data structures, i.e. to compute an array element only the values of neighboring elements are needed. When updating the boundary conditions some non-local operations are applied, such as copying plane 91 onto plane 1 and plane 2 onto plane 92, thus simulating the closed crucible.

The code is designed such that a simulation can be split into a sequence of program runs. The program generates a continuation data set at the end of a run. In addition, data can be output during the time loop to allow off-line

visualization. The I/O operations could not be parallelized by both tools and are only supported in the manual version.

3 Parallelization Strategy

The code is parallelized according to the data partitioning approach. The main arrays are divided into blocks that are assigned to the processors. For the application the HPF BLOCK distribution strategy is optimal. Depending on the number of processors all three dimensions can be partitioned.

The computation for such a distributed array is spread among the processors with respect to data locality. Most systems apply the owner-computes rule, i.e. the owner of an array element computes its new values. Current parallelization tools either strictly apply this rule or individual loops are spread according to the owner of a single array reference in the loop body and assignments to other arrays may be executed for array elements not owned by the processor.

The transformation of Fortran 77 with data distribution annotations to message passing code consists of several steps: interprocedural distribution analysis, computation of communication patterns according to the data distribution and the resulting work distribution, shrinking of arrays to save memory in each processor, and implementation of the communication with as few synchronizations as possible. The techniques applied in these steps are well developed for regular computations but are still in a research stage for irregular applications like finite-element codes.

In this application, the distribution of arrays in blocks leads to two communication patterns: an overlap update and a remote copy. If we assume a distribution in the second dimension, the stencil operations induce access to the boundary elements of the neighboring blocks. Therefore, the left- and rightmost plane have to be sent to the neighboring processors prior to the computation.

The copy operation outlined in the previous section leads to communication between the rightmost and the leftmost processor only if the second dimension is distributed. In all other cases this operation can be executed locally. There is a similar operation to be performed in the first dimension.

4 Vienna Fortran Compilation System

The VFCS developed at the University of Vienna is the successor of the SUPERB system. The input language is Vienna Fortran, a language extension to Fortran 77 providing very flexible data distribution mechanisms.

The system performs an automatic communication analysis which is specifically tailored to overlap communication resulting from stencil operations. The tool computes the overlap areas automatically from the work distribution and the data distribution of right-hand side arrays.

Since the VFCS applies the owner-computes rule, assignments to auxiliary variables in loops are executed by all processors, i.e. scalar variables are replicated. If such an assignment is part of a stencil operation the overlap area cannot

be determined precisely. In such situations the user can apply several transformations to interactively optimize the code.

Mask propagation [4] determines, on the basis of dataflow information, in which iterations a processor has to execute such an assignment. If those iterations are unique the information can be used to determine the communication pattern. This transformation is very useful for our application and makes it possible to detect most of the stencil operations.

Since the VFCS is tailored to overlap communication, copy operations in that application cannot be handled efficiently. They lead to an overlap area in each process that consists of the total array. Since this results in a lot of communication and memory overhead only the third dimension of the arrays was distributed. In that dimension no copy operations occur.

In the communication optimization step the communication is vectorized and performed very efficiently. An important missing optimization is the combination of messages updating overlap areas of different arrays. Although the messages are exchanged among the same pair of processors they are not combined to one message.

During the work distribution optimization the loops are transformed such that processors only execute those iterations that contain local computations. This transformation is successful for almost all loops in that program. Only in a few cases the distribution of work is performed on statement level. Individual masks for statements in a loop are generated in the form of logical-ifs because the loop transformation cannot ensure the owner-computes rule. A simple extension of the existing transformation that allows to mask the entire loop based on loop invariant expressions used in the statement-masks would allow to generate efficient code.

The tool automatically shrinks the arrays to an appropriate shape for the local part and the overlap area, i.e. the memory where communicated overlap information is stored. This transformation as well as some other transformations imply that the number of processors and the distribution is fixed at compile-time. The generated code uses very few functions of a runtime library and thus can be optimized manually.

5 FORGE 90

FORGE 90 is a commercial parallelization tool developed by Applied Parallel Research. Similar to the VFCS it is an interactive tool that supports data distributions as the basis for parallelization. The tool currently supports one-dimensional block and cyclic distribution.

In the first step, the user supplies the data distribution. Then the user has to pick individual loops in the program to spread the iterations over the processors. In contrast to the VFCS that strictly applies the owner-computes rule, the generated code may contain remote stores.

The user can explicitely specify a work distribution which gives him much flexibility to optimize data locality. When picking loops the user has to take

into account that FORGE 90 will implement communication resulting from the data distribution and the work distribution in the best case before and after the selected loop. Thus messages are vectorized with respect to this loop but more aggressive message vectorization across surrounding sequential loops is not supported.

This enforces that the first dimension of the arrays in our application has to be distributed. The loops are implemented in such a way that the longest loop, i.e. running over the second dimension, is the innermost to allow efficient vectorization on Cray. Manual loop interchange in the entire code is an error-prone operation and communicating values in the inner loops leads to very small messages and a lot of communication overhead.

The loop partitioning step computes information which values have to be communicated in a loop. The information presented is imprecise since it consists of all references to distributed arrays. An analysis whether communication is really necessary is later on performed in the backend. But an experienced user can use the presented information to determine those references that do not need any communication and thus can help the backend to eliminate unnecessary runtime overhead.

Shrinking of arrays can be specified with the data distribution. Since memory for shrinked arrays is allocated dynamically and all references to shrinked arrays are linearized, the performance is not as good as with full-size arrays. Although the difference was within a few percent we used full-size arrays for our performance measurements. Due to these memory allocation strategies the generated code can be executed on any number of processors. It includes a lot of calls to a runtime library and thus manual optimization of the generated code is nearly impossible.

6 Conclusion

Our experiments on the iPSC/860 showed that both tools lead to similar performance if the data distribution was selected with respect to the capabilities of the tools, i.e. distribution of the third dimension for the VFCS and of the first dimension for FORGE 90 [1]. Both tools support only one-dimensional distributions for this application. The one-dimensional hand-coded version is 10 percent faster than the automatically parallelized versions (Table 1).

The HPF BLOCK distribution is not suited if the number of elements in a dimension is not much larger than the number of nodes. For example, the 16-node version of Manual1D can utilize only 14 nodes and the 32-node version 21 nodes. The Vienna Fortran BLOCK strategy distributes the 42 elements in the third dimension more evenly among the processors.

The two-dimensional hand-coded version performed much better due to a lower communication overhead and a better load balance. In future, similar execution times can be expected for the automatically parallelized code if the

[1] The FORGE 90 test version was limited to 16 nodes.

processors	FORGE 90	VFCS	Manual1D	Manual2D
1	198	197	176	176
2	120	116	104	100
4	79	82	70	50
8	58	55	54	29
16	41	42	37	17
32	*	39	35	10.5
speedup 16	4.8	4.7	4.8	10.5
speedup 32		5	5	17

Table 1. Execution Times (secs) and Speedup

tools are able to handle multi-dimensional distributions effectively. Extensions of the techniques used in both tools, i.e. the compile-time analysis as well as the code generation strategies for communication and work distribution, would be sufficient for efficient parallelization of that application.

Acknowledgement

We thank K. Wingerath (Institut für Festkörperforschung) for providing access to the application and giving us a lot of background information and R. Dissemond for implementing the manually parallelized version.

References

1. U. Detert, H.M. Gerndt, *TOP^2 : Tool Suite for Partial Parallelization, Version 2.01, User's Guide*, Forschungszentrum Jülich, Interner Bericht KFA-ZAM-IB-9321, 1993
2. Applied Parallel Research, *FORGE 90, Distributed Memory Parallelizer, User's Guide, Version 8.7, User's Guide*, 1993
3. S. Hiranandani, K. Kennedy, C. Tseng, *Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines*, Proceedings of the Supercomputing Conference 1991, Albuquerque, 86-100, November 1991
4. M. Gerndt, *Updating Distributed Variables in Local Computations*, Concurrency: Practice and Experience, Vol. 2(3), 171-193, September 1990
5. HPFF, *High Performance Fortran Language Specification*, High Performance Fortran Forum, Version 1.0, Rice University Houston Texas, May 1993
6. H. Zima, P. Brezany, B. Chapman, P. Mehrotra, A. Schwald, *Vienna Fortran - A language Specification Version 1.1*, University of Vienna, ACPC-TR 92-4, March 1992
7. M. Mihelcic, H. Wenzl, K. Wingerath, *Flow in Czochralski Crystal Growth Melts*, Bericht des Forschungszentrums Jülich, No. 2697, ISSN 0366-0885, December 1992

This article was processed using the LaTeX macro package with LLNCS style